Capstone Project Proposal: Nola Local a Hyper-Local Event Discovery Platform for New Orleans

1. What tech stack will you use for your final project?

Frontend:

- **React** with **Next.js** (App Router for routing and page structure)
- **TypeScript** for type safety
- **Redux Toolkit** for state management (user auth state, event filters, calendar state, UI state)
- **Tailwind CSS** for styling
- **Framer Motion** for animations and transitions
- **React Hook Form** + **Zod** for form validation

UI Libraries & Assets:

- **Lucide React** for icons (free, modern icon library)
- **date-fns** or **Day.js** for date manipulation
- **React Big Calendar** or custom calendar component for calendar view
- **Unsplash API** (free tier) for placeholder event images if users don't upload their own

Backend (MVC Architecture):

- **Next.js API Routes** for backend endpoints
- **Node.js** runtime environment
- **MVC Pattern Implementation**:
    - **Models**: Mongoose schemas (User, Event, Category)
    - **Views**: React components (frontend)
    - **Controllers**: API route handlers in /app/api/ directory that handle business logic, interact with models, and return responses

Database:

- **MongoDB Atlas** (free tier, cloud-hosted NoSQL database)
- **Mongoose** for object modeling and schema definitions

Authentication:

- **NextAuth.js** with JWT strategy
- **bcrypt.js** for password hashing

Email Services:

- **Nodemailer** for sending emails
- **Mailtrap** (free tier) for development/testing email delivery
- **Resend** or **SendGrid** (free tier) for production email delivery

External APIs & Data Management:
- **Eventbrite API** (free tier) for bootstrapping initial event data
- **Ticketmaster Discovery API** (free tier) as secondary event source
- Custom sync service/controller to periodically fetch and transform external event data

Image Storage:
- **Cloudinary** (free tier) for user-uploaded event images

Deployment:
- **Vercel** for hosting (free tier, optimized for Next.js)

Typography (inspired by Meatpacking's bold aesthetic):
- **Primary**: Inter or Manrope (modern, clean sans-serif for body text)
- **Display**: Space Grotesk or Clash Display (bold, contemporary for headlines)
- Both available via Google Fonts

2. Is the front-end UI or the back-end going to be the focus of your project?

This will be an **evenly focused full-stack application** following **MVC architecture**.

Backend Focus (Controllers & Models):
- Robust Mongoose models for users, events, categories
- Controller logic for external API data transformation pipeline (fetch → clean → normalize → store)
- Authentication controllers with JWT-based protected routes and email verification
- Complex filtering and query logic in event controllers
- CRUD controllers for user-generated events
- Middleware for authorization checks
- Email verification system with token generation and validation

Frontend Focus (Views):
- Redux Toolkit for centralized state management (auth slice, events slice, filters slice, UI slice)
- Multiple interactive views (calendar grid, event cards, detail modals)
- Real-time filtering and search functionality
- Responsive design with bold, modern aesthetic inspired by Meatpacking District
- Smooth animations and transitions
- Email verification flow UI

3. Will this be a website? A mobile app? Something else?

This will be a **responsive website** optimized for modern browsers, built as a **Single-Page Application (SPA)** using Next.js's App Router with both server-side and client-side rendering capabilities.

4. What goal will your project be designed to achieve?

The goal is to solve the problem of fragmented local event information in New Orleans by creating a **centralized, interactive platform** where users can:

- Discover current and upcoming local events from multiple sources
- Create and share their own grassroots events
- Filter and search events by date, category, and source
- Build a personalized experience through liked/saved events
- Connect with the local community through verified accounts

This addresses the problem of always learning about New Orleans events too late and creates value for both event-goers and event creators in the local community.

5. What kind of users will visit your app?

Primary Demographics:

- **Local Residents (Ages 21-45)**: Active individuals interested in discovering food, arts, music, and community events in New Orleans
- **Event Creators/Small Businesses**: Local musicians, artists, pop-up vendors, and small businesses looking for a free platform to promote their events
- **Tourists/Visitors**: People visiting New Orleans who want to discover authentic local happenings beyond typical tourist attractions

6. Data and API Approach

Data Sources & Collection Strategy:

External API Integration (for initial population):

- **Eventbrite API**: Primary source for structured event data
- **Ticketmaster Discovery API**: Secondary source for concerts, sports, entertainment

Data Transformation Pipeline (Controller Layer):
A dedicated ExternalEventController will handle:

1. **Fetch**: Scheduled sync (daily cron job via Vercel Cron or manual trigger endpoint) fetches events from external APIs filtered by:
    - Location: New Orleans, LA (or within radius)
    - Date range: Today + 90 days forward

Transform: Controller normalizes external data to match internal Event model:

```
// Example transformation in controller
```

```
const transformEventbriteEvent = (externalEvent) => ({
  title: externalEvent.name.text,
  description: externalEvent.description.text,
  date: new Date(externalEvent.start.local),
  time: extractTime(externalEvent.start.local),
  location: externalEvent.venue.address.localized_address_display,
  category: mapToCategory(externalEvent.category_id),
  imageUrl: externalEvent.logo?.url || null,
  source: 'eventbrite',
  sourceUrl: externalEvent.url,
  externalId: externalEvent.id,
});
```

2. **Store**: Controller saves to MongoDB via Event model with:
   Deduplication logic (check externalId + source)
   - Attribution metadata for proper credit
   - Auto-archive events after their date passes (via scheduled cleanup controller)

User-Generated Data:
- Users create original events through authenticated forms
- EventController.create() validates, processes, and stores with source: 'user' and creator: userId

Image Storage:
- User-uploaded images stored in Cloudinary (not in MongoDB)
- MongoDB stores only the Cloudinary URL string
- External API events already provide image URLs that are saved directly

Attribution Strategy:
- Event cards display source badge ("via Eventbrite")
- Source name links to original event page via sourceUrl
- Complies with API Terms of Service for attribution

Project Approach

1. Database Schema (Models)

User Model (models/User.js):

```
{
  _id: ObjectId,
  username: String (unique, required, indexed),
  email: String (unique, required),
  password: String (hashed, required),
  isVerified: Boolean (default: false),
  verifyToken: String,
```

```
  verifyTokenExpiry: Date,
  resetToken: String, // for stretch goal
  resetTokenExpiry: Date, // for stretch goal
  createdAt: Date,
  createdEvents: [ObjectId], // references Event
  likedEvents: [ObjectId], // references Event
}
```

Event Model (models/Event.js):

```
{
  _id: ObjectId,
  title: String (required, indexed),
  description: String (required),
  date: Date (required, indexed),
  time: String,
  location: String (indexed),
  category: String (enum: ['Live Music', 'Food & Drink', 'Arts & Culture', 'Community', 'Sports',
'Other'], indexed),
  imageUrl: String, // Cloudinary URL or external API URL
  source: String (enum: ['user', 'eventbrite', 'ticketmaster'], indexed),
  sourceUrl: String, // external link or null for user events
  externalId: String, // for deduplication, null for user events, indexed
  creator: ObjectId (references User, null for external events),
  likes: [ObjectId], // references Users who liked
  status: String (enum: ['upcoming', 'passed'], indexed),
  createdAt: Date,
}
```

Category Model (models/Category.js) - pre-seeded:

```
{
  _id: ObjectId,
  name: String (required),
  slug: String (unique, indexed),
  color: String, // hex color for UI
}
```

2. MVC Architecture Structure

```
/app
  /api                        // Controllers (API Routes)
    /auth
      /login/route.js          // AuthController.login
      /signup/route.js          // AuthController.signup
      /logout/route.js         // AuthController.logout
      /verifyemail/route.js    // AuthController.verifyEmail
```

```
    /events
      /route.js              // EventController.getAll, create
      /[id]/route.js         // EventController.getById, update, delete
      /like/route.js         // EventController.toggleLike
      /sync/route.js         // ExternalEventController.syncEvents
    /users
      /[id]
        /events/route.js     // UserController.getCreatedEvents
        /likes/route.js      // UserController.getLikedEvents
    /upload/route.js         // ImageController.upload (Cloudinary)

/models                      // Models (Mongoose schemas)
  User.js
  Event.js
  Category.js

/lib                         // Business logic helpers
  /controllers               // Controller helper functions
    authController.js
    eventController.js
    externalEventController.js
    userController.js
    emailController.js
  /middleware
    authMiddleware.js        // JWT verification
    errorHandler.js
  /utils
    apiTransformers.js       // External API data transformation
    validation.js
    cloudinary.js            // Cloudinary config

/store                       // Redux Toolkit setup
  store.js                   // Configure store
  /slices
    authSlice.js             // User auth state
    eventsSlice.js           // Events data, filters
    uiSlice.js               // Modals, loading states

/components                  // Views (React components)
  /layout
    Navbar.js
    Footer.js
  /events
    EventCard.js
```

```
    EventModal.js
    EventForm.js
    CalendarView.js
    EventGrid.js
  /auth
    LoginForm.js
    SignupForm.js

/app
  /verifyemail
    page.tsx                // Email verification landing page
```

3. Redux Toolkit State Management

Auth Slice (/store/slices/authSlice.js):

```
{
  user: { id, username, email, isVerified } | null,
  token: string | null,
  isAuthenticated: boolean,
  loading: boolean,
  error: string | null
}
```

Events Slice (/store/slices/eventsSlice.js):

```
{
  events: Event[],
  selectedEvent: Event | null,
  filters: {
    category: string | null,
    dateRange: { start: Date, end: Date } | null,
    source: 'all' | 'user' | 'eventbrite' | 'ticketmaster',
    searchQuery: string
  },
  view: 'calendar' | 'grid',
  loading: boolean,
  error: string | null
}
```

UI Slice (/store/slices/uiSlice.js):

```
{
  isEventModalOpen: boolean,
  isCreateEventModalOpen: boolean,
  isMobileMenuOpen: boolean,
  toast: { message: string, type: 'success' | 'error' | 'info' } | null
}
```

4. What kinds of issues might you run into with your API?

External API Challenges:

- **Rate Limits**: Free tiers have request limits (e.g., Eventbrite: 1000 requests/hour). **Solution**: Cache responses in database, sync daily instead of real-time, implement request throttling in controller
- **Data Inconsistencies**: Different APIs structure data differently. **Solution**: Build robust transformation layer in externalEventController.js with error handling, data validation, and fallback values
- **API Downtime**: External services may be unavailable. **Solution**: Implement retry logic with exponential backoff, graceful degradation (show user-created events only if external APIs fail), error logging
- **Duplicate Events**: Same event may appear on multiple platforms. **Solution**: Deduplication logic in controller based on fuzzy matching (title similarity + date + location proximity)
- **Stale Data**: External events may be cancelled/updated. **Solution**: Re-sync daily, store lastSyncedAt timestamp, allow users to report outdated events

Internal API Challenges:

- **N+1 Query Problem**: Fetching events + creator + likes requires multiple DB queries. **Solution**: Use MongoDB aggregation pipelines or Mongoose .populate() with .select() to limit fields
- **Timezone Handling**: Ensuring consistent date/time storage and display. **Solution**: Store all dates in UTC in database, convert to Central Time (New Orleans) in controllers before sending to client
- **Authorization**: Ensuring users can only edit/delete their own events. **Solution**: Middleware in controllers checks event.creator.toString() === req.user.id before allowing modifications
- **Complex Filtering**: Multiple simultaneous filters (date range, category, source, search query). **Solution**: Build dynamic MongoDB query object in controller based on provided filters

Email-Related Challenges:

- **Token Expiration**: Verification links should expire (e.g., 24 hours). **Solution**: Store verifyTokenExpiry in database, check verifyTokenExpiry > Date.now() on verification
- **Email Delivery**: Emails may land in spam or fail to send. **Solution**: Use reputable service (Mailtrap for dev, Resend/SendGrid for production), implement proper SPF/DKIM records in production
- **Special Characters in Tokens**: bcrypt hashes may contain URL-unsafe characters. **Solution**: URL-encode tokens before embedding in links, decode on verification
- **Development Testing**: Don't want to spam real emails during development. **Solution**: Use Mailtrap to catch all emails in development environment

Image Storage Challenges:

- **File Size**: Large images slow down uploads and consume storage. **Solution**: Cloudinary automatic optimization and transformations
- **Upload Failures**: Network issues or service downtime. **Solution**: Implement retry logic, allow events without images, provide error feedback to users

5. Is there any sensitive information you need to secure?

Yes:

- **MongoDB Connection String** (MongoDB Atlas credentials)
- **NextAuth Secret** and **JWT Secret Key** for token signing
- **External API Keys** (Eventbrite API key, Ticketmaster API key)
- **Cloudinary API credentials** (cloud name, API key, API secret)
- **Email Service Credentials** (Mailtrap/Resend/SendGrid API keys)

Security Strategy:

- Store all secrets in .env.local file (excluded from Git via .gitignore)
- Access via process.env.VARIABLE_NAME in Next.js
- For production deployment: Configure environment variables in Vercel dashboard
- Never commit or expose secrets in client-side code
- Use server-side API routes (controllers) to interact with external APIs
- Email service credentials only used in server-side controllers

6. Functionality / User Flow

Feature	Description	Controller	Protected?
Landing Page	Public view showing featured/random events carousel (read-only)	EventController.getFeatured	Public
Authentication	Sign up, Log in (NextAuth with JWT)	AuthController.signup, .login	Routes /login, /signup redirect if logged in
Email Verification	Upon signup, user receives verification email with token	EmailController.sendVerificationEmail	Triggered automatically

Verify Email Page	User clicks link in email, lands on verification page that validates token	AuthController.verifyEmail	Public (token-based)
Resend Verification	User can request new verification email if expired	EmailController.resendVerification	Public
Event Discovery	Main dashboard with calendar view, card grid, filters, search	EventController.getAll with query params	**Protected**
Event Detail Modal	Full details, source attribution, like button, external link	EventController.getById	**Protected**
Event Creation	Form: title, description, date, time, location, category, image upload	EventController.create, ImageController.upload	**Protected**
Event Interactions	Like/unlike events	EventController.toggleLike	**Protected**
Event Management	Edit/delete own created events	EventController.update, .delete	**Protected** + Authorization
User Dashboard	"My Created Events" and "My Liked Events" tabs	UserController.getCreatedEvents, .getLikedEvents	**Protected**
Search & Filter	Date range, category, source type, text search	EventController.getAll with filters	**Protected**

External Sync	Admin/cron endpoint to sync external events	ExternalEventController.syncEvents	**Protected** (admin or cron only)

Detailed User Flow:

1. **First-Time Visitor**:
    - Lands on homepage → sees animated event carousel (public, read-only) via EventController.getFeatured
    - Clicks "Sign Up" → fills form → AuthController.signup creates account with isVerified: false
    - **EmailController.sendVerificationEmail automatically sends verification email with hashed token**
    - **User receives email, clicks link → lands on /verifyemail?token=xxx**
    - **Frontend page extracts token from URL, calls AuthController.verifyEmail → if valid, sets isVerified: true**
    - **Success message displayed, user can now log in**
    - User logs in via AuthController.login → JWT issued → redirected to calendar view

2. **Returning User**:

    - Logs in via AuthController.login → must be verified (isVerified: true) to access protected routes → JWT issued
    - Redux stores auth state, full calendar/grid view loads via EventController.getAll
    - Uses filters → Redux updates filter state → triggers new API call with query params → EventController.getAll returns filtered results
    - Clicks event → Redux opens modal, loads data from EventController.getById
    - Clicks like → EventController.toggleLike updates database → Redux updates local state optimistically
    - Clicks "Create Event" → Modal opens → fills form → uploads image to ImageController.upload (Cloudinary) → gets URL → EventController.create stores event with Cloudinary URL

3. **Event Creator**:
    - Creates event via EventController.create with source: 'user'
    - Can edit via EventController.update (authorization middleware checks ownership)
    - Can delete via EventController.delete (authorization middleware checks ownership)
    - Views their events in dashboard via UserController.getCreatedEvents

Email Verification Flow Details:

1. **On Signup**:

- AuthController.signup creates user with isVerified: false
- Generates hashedToken using bcrypt.hash(userId.toString())
- Stores hashedToken and verifyTokenExpiry (24 hours from now) in User document
- EmailController.sendVerificationEmail sends email via Nodemailer with link: https://localpulse.com/verifyemail?token={hashedToken}

2. **User Clicks Link**:
   - Frontend page (/verifyemail) uses useEffect to extract token from window.location.search
   - Makes POST request to /api/auth/verifyemail with token in body
   - Backend finds user with matching verifyToken where verifyTokenExpiry > Date.now()
   - If valid: sets isVerified: true, clears verifyToken and verifyTokenExpiry, saves user
   - Returns success response → frontend displays success message

3. **Login Protection**:
   - AuthController.login checks isVerified status before issuing JWT
   - Unverified users cannot log in (returns error: "Please verify your email")
   - Option to resend verification email via EmailController.resendVerification

7. What features make your site more than a CRUD app?

Beyond Basic CRUD:

1. **External API Integration Pipeline**:
   - Automated sync controller (ExternalEventController) that fetches from multiple APIs
   - Data transformation and normalization logic
   - Deduplication algorithm
   - Source attribution system
   - Error handling and retry logic

2. **Complex Filtering & Search**:
   - Multi-dimensional filtering in EventController (date range, category, source, status)
   - Real-time text search across titles and descriptions using MongoDB text indexes
   - Calendar view with interactive date selection
   - Dynamic query building based on multiple parameters
   - Redux manages filter state and triggers API calls

3. **Interactive Calendar Interface**:
   - Custom calendar component showing aggregated events by date
   - Toggle between calendar and card grid views (managed by Redux)
   - Date range selection affecting controller queries

4. **Centralized State Management with Redux Toolkit**:
    - Auth slice managing user session, token, verification status
    - Events slice managing event data, filters, search, view preferences
    - UI slice managing modals, toasts, loading states
    - Optimistic updates for likes
    - Persisting filter preferences in localStorage via Redux middleware

5. **Email Verification System**:
    - Token generation and hashing with bcrypt
    - Secure email delivery with Nodemailer
    - Token expiration and one-time use validation
    - Frontend verification flow with URL parameter extraction
    - Email template system

6. **Authorization Layer**:
    - Middleware protecting routes and API endpoints
    - Ownership verification in controllers before allowing modifications
    - Role-based access (admin endpoints for external sync)
    - Email verification requirement for login

7. **Image Upload & Processing**:
    - Integration with Cloudinary via dedicated controller
    - Image optimization, resizing, and transformations
    - Handling both user uploads and external API images
    - MongoDB stores only URLs, not binary data

8. **Scheduled Jobs**:
    - Daily cron job triggering ExternalEventController.syncEvents
    - Automated cleanup of past events
    - Periodic deduplication checks

9. **Responsive, Animated UI**:
    - Smooth transitions with Framer Motion
    - Modal system for event details
    - Loading states and skeleton screens
    - Bold, modern design with custom color palette inspired by Meatpacking District

8. Stretch Goals

- **Password Reset Flow**: Apply same token/email pattern for "forgot password" feature (models already support resetToken and resetTokenExpiry)
- **Welcome Email**: Send personalized welcome email after verification
- **Email Notifications**: Notify users when their created events receive likes or when events they liked are happening soon
- **Advanced Search**: Full-text search with MongoDB Atlas Search or Algolia integration

- **Social Sharing**: Controller generates shareable Open Graph images for events
- **User Profiles**: Public profiles showing created events, reputation/follower system
- **Event Check-ins**: Users can mark attendance, stored in Event model
- **Recommendation Engine**: Controller analyzes liked/attended history to suggest events using Redux state
- **Admin Dashboard**: Moderation tools, analytics, manual event approval system
- **Comments/Reviews**: Users can leave feedback on events they attended
- **Geographic Filtering**: Map-based discovery (original concept stretch goal)